WHISPER: Wirelessly synchronized distributed audio sensor platform

Kiselev, Ilya ; Ceolini, Enea ; Wong, Daniel ; de Cheveigné, Alain ; Liu, Shih-Chii

Abstract: This paper describes a distributed wireless acoustic sensor network (WASN) platform called WHISPER that is capable of synchronous multichannel sampling at different spatial locations with a sampling clock whose relative jitter is less than 300 ns. The platform comprises up to four data acquisition modules with onboard computing capabilities, and that can form an ad-hoc Wi-Fi network allowing an additional processing module such as a laptop or a smartphone to be connected if needed. Each acquisition module holds four digital SPI microphones with a total of 16 microphones for the entire system. Wireless synchronization of the sampling on each platform is implemented using a separate wireless module operating in the 902-928 MHz ISM band. Usage of this system is demonstrated in a real-time application involving spatial sound filtering through a beamforming algorithm.

DOI: https://doi.org/10.1109/LCN.Workshops.2017.62

Accepted for publication in IEEE 42nd Conference on Local Computer Networks Workshops (LCN Workshops)

WHISPER: Wirelessly Synchronized Distributed Audio Sensor Platform

Ilya Kiselev[1], Enea Ceolini[1], Daniel Wong[2], Alain de Cheveigne[2], and Shih-Chii Liu[1]

[1]Institute of Neuroinformatics
University of Zurich and ETH Zurich,
Zurich, Switzerland
kiselev, eceoli, shih@ini.uzh.ch

[2]Laboratoire des systèmes perceptifs, DEC, ENS, PSL
Research University, CNRS
Paris, France
daniel.wong, alain.de.cheveigne@ens.fr

***Abstract*—This paper describes a distributed wireless acoustic sensor network (WASN) platform called WHISPER that is capable of synchronous multichannel sampling at different spatial locations with a sampling clock whose relative jitter is less than 300 ns. The platform comprises up to four data acquisition modules with onboard computing capabilities, and that can form an ad-hoc Wi-Fi network allowing an additional processing module such as a laptop or a smartphone to be connected if needed. Each acquisition module holds four digital SPI microphones with a total of 16 microphones for the entire system. Wireless synchronization of the sampling on each platform is implemented using a separate wireless module operating in the 902–928 MHz ISM band. Usage of this system is demonstrated in a real-time application involving spatial sound filtering through a beamforming algorithm.**

Keywords—wireless acoustic sensor network, beamforming, audio platform synchronization, distributed network, wireless synchronization

I. INTRODUCTION

Increasing the signal-to-noise (SNR) of target audio sources is useful for applications such as hearing aid devices that have only two to four microphones that are spaced closely together; and ambient intelligent space monitoring. Although the SNR of a source can be improved acoustically by placing the microphone next to the source or by fixed microphone arrays, a distributed ad-hoc network of microphones provides the most flexible solution as in a distributed wireless acoustic sensor network (WASN).

In general, a WASN that supports as many distributed microphones as possible within a space will allow more spatially coherent sources to be isolated with shorter impulse responses [1], and better rejection of diffuse noise [2]. It is also more likely that at least one microphone will be close to the target or to a major noise source that needs to be factored out.

Computation capabilities should be added at each node so that bandwidth usage is reduced and the platform can support distributed processing algorithms. Each node would then transmit a smaller number of processed signals instead of transmitting all its microphone signals simultaneously to the processing node. It also means that the network can be scaled up by adding more nodes without a noticeable impact on the overall computational load and bandwidth usage of the system [3].

With local computing power, a node can perform within-node processing on the signals of their own microphones, and possibly together with signals received from neighboring nodes. A node that "sees" a source with the best SNR could "own" the source [4] therefore allowing the source to be cleaned before it is made available for other nodes. Within-node processing can also benefit from signals transmitted from those nodes that own the noise source.

Other forms of within-node preprocessing include inter-stream correlation that achieves a degree of compression of the signals and furthermore, once a filter solution has been computed, each node then needs to transmit only one stream to others or to the hearing aid [5]. Thus, the network can deliver to the user's ears a signal with a better SNR than any individual microphone.

With current technology, one can construct a portable platform with enough computational power for the aforementioned local processing by using devices such as FPGAs and embedded computers. Such a platform can then be used for prototyping real-time sound processing algorithms such as spatial sound filtering, noise cancellation and blind source separation. Some of these algorithms require synchronized microphone samples from the network, therefore one of the main challenges in building a useable WASN platform is the issue of sampling clock synchronization.

Prior work in clock synchronization across multiple nodes of a distributed multi-microphone network includes the realigning of the clocks based on acoustic cues, wireless synchronization, GPS, or by blind synchronization techniques [6][7][8][9]. One study tested the latter method in an off-line multi-talker speech recognition task using an ad hoc network of smartphones and the cloud [10]. The solutions in [7][8][9] reported synchronization precision in order of tens of microseconds but none of them address low-latency data transmission. This work addresses both the low-latency data transmission requirements and the sampling clock synchronization requirements across multiple nodes in a WASN.

This work was partially supported by the European Union's Horizon 2020 research and innovation programme under grant agreement No 644732 and the Swiss National Foundation Research Project FRAS #200020-153565.

We call our multi-microphone platform WHISPER, which stands for "Wireless Hearing Improves Speech PERception". The platform provides a testbed for developing and testing source SNR improvement algorithms which could improve speech intelligibility for hearing aid users, but it also can be used for testing other audio processing algorithms which require synchronous distributed sampling. We present the challenges of building this platform in Section II, the platform details in Section III, the multi-microphone synchronization method in Section IV, a beamforming algorithm in Section V, results from this algorithm applied to the wirelessly transmitted microphone samples from two platform modules in Section VI and finally, conclusion of the work in Section VII.

II. Challenges of Platform Construction

The development of the WHISPER platform focuses on the task of real-time spatial filtering with the aim of possible use for speech enhancement with hearing aid devices. The platform is composed of four spatially separated modules which are placed across a room and which are connected through a wireless network. This design imposes the following challenges involving sampling clock synchronization, synchronization of packets received wirelessly from different modules, and wireless communication latency.

Sampling clock synchronization across modules is an issue when processing audio streams from nodes with different oscillators, because the time alignment between the streams is unknown, and even drifts with time. Synchronous microphone sampling on different modules is important for algorithms such as blind source separation, where previous work showed significant drops in performance with non-synchronized microphone samples [11]. Synchronization is also crucial for source localization and segregation based on sensor array geometry. The work in [7] reported a solution for time synchronization of 10 μs precision across multiple nodes, but the platform does not provide sampling clock synchronization. The work in [8] uses a wireless sampling clock synchronization method similar to the approach proposed in this work, but their method based on a sub-GHz ISM transmitter and a subsequent Frequency Locked Loop (FLL) provides only 20 μs precision in the sampling phase synchronization. The work in [9] described a platform with the sampling clock synchronization across the nodes, but without specifying any numbers.

Due to the low-latency requirements of the intended platform, we cannot apply the blind synchronization technique proposed in [6]. It is also impossible to use either NTP (Network Time Protocol) or GPS (Global Positioning System) signals for the sampling clock synchronization because NTP does not provide sufficient accuracy and GPS is not available inside buildings.

Latency of wireless transmission is also a critical factor. Ideally, transmitted signals should arrive no later than propagated acoustic signals to avoid a time difference between the transmitted sound and the direct sound. This is important for people who have mild hearing loss at lower frequencies and use non-occluding hearing aids [12]. If the transmitted signal arrives with a delay, acoustically transmitted noise cannot be perfectly cancelled. Constraints of wireless technology make this ideal of faster-than-sound transmission difficult to achieve, and therefore one may need to settle for the lesser goal of minimizing the perceptual mismatch between acoustic and visual cues.

The three challenges and solutions proposed for this platform are described below:

1) Synchronization of sampling clock frequency and phase across different modules of the platform. Our solution is presented in Section III.A and Section IV.A.

2) Synchronization of wirelessly transmitted data packets at the processing module. We investigated the use of Wi-Fi for wireless transmission of data between modules. Because of its asynchronous transmission manner and unpredictable data delivery latency, a packet level synchronization solution is needed, which is presented in Section IV.B.

3) Minimization of transmission latency of audio data packets between the modules of the platform. Measurements of the transmission latencies using Wi-Fi (presented in Section IV.A) show that another solution for wireless data transmission is needed. This solution is presented in Section III.C.

III. Four-microphone WHISPER-M4 Module

The distinct features of the WHISPER platform are: (i) its wireless modular design; (ii) synchronous sampling of multiple spatially distributed microphones; (iii) local processing capabilities (FPGA+software); (iv) potentially low latency of data transmission; (v) relatively low power consumption; and (vi) extension possibilities. The WHISPER platform can accommodate up to four WHISPER-M4 modules that will be described further in the following subsections.

A. Hardware

One of the advantages of this platform is that it is constructed out of only commercially available off-the-shelf (COTS) components, making it ideal for rapid prototyping because there is no development of any custom hardware. The block diagram of a WHISPER-M4 module is depicted in Fig. 1 with a picture shown in Fig. 2.

The platform module has a computing unit (Raspberry Pi 3 Model B) and an FPGA LOGI-PI-2 board from ValentF(x) (http://valentfx.com/logi-pi) that is used to implement the synchronization algorithm and to extend the IO capabilities of the Raspberry Pi board. It supports synchronous sampling of up to four SPI microphones now, but it is also possible to incorporate any other type of sensor with a digital interface. The FPGA can also be used to do some signal pre-processing, such as digital filtering, down sampling, automatic gain control, and post-processing, such as compression, scrambling and error-correction code.

To implement wireless synchronization of data sampling across different modules of the platform, as described in challenge 1) in Section II, we use the MRF89XAM9A transceiver from Microchip Technology Inc. operating in ISM 902–928 MHz frequency band. The synchronization algorithm implemented using this transceiver is described in Section IV.A. The transceiver can also be used for unidirectional low-

bandwidth data broadcasting from the synchronization master to all other modules.

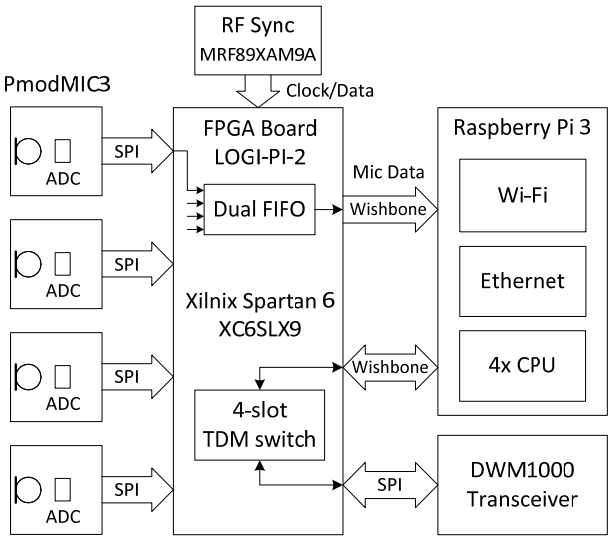

Fig. 1. Block diagram of the WHISPER-M4 module.

There are three options for wireless connectivity: 2.4 GHz Wi-Fi (IEEE 802.11n), Bluetooth 4.1 and ultra-wideband 3.5-6.5 GHz transceiver DWM1000 (IEEE 802.15.4). Wi-Fi and Bluetooth use the Broadcom BCM43438 chip integrated in the Raspberry Pi 3 model B. DWM1000 is an external module from DecaWave connected to FPGA via SPI bus (Fig. 1). The full specifications are given in TABLE I.

B. *FPGA Logic and Software for Data Acquisition*

The FPGA is connected to the Raspberry Pi via a high-speed SPI bus. At the software level, the Wishbone interface is used, which is based on the standard Wishbone libraries provided with the LOGI-PI-2 board. The maximal required data throughput of this interface for 4 microphones sampled at 48 kHz and 16 bit resolution is 3.072 Mbit/s. In order to minimize the communication overhead and to achieve such a data rate, we implemented a dual FIFO in the FPGA and used the burst mode of the Wishbone interface (see TABLE II for the address space). Using the burst mode helps to avoid sending an address every time we need to read a data word, but introduces some latency due to the time needed to fill the buffer.

TABLE I. WHISPER-M4 MODULE SPECIFICATIONS

CPU	Broadcom BCM2837 1.2GHz 64-bit quad-core ARMv8
RAM	1GB LPDDR2 @ 900 MHz
FPGA	Xilinx Spartan 6 XC6SLX9TQG144 9152 Logic Cells, 200 User IO Pins
Wireless Interfaces	2.4 GHz Wi-Fi (802.11n), 150Mbits/s Bluetooth 4.1 (BLE), 24 Mbits/s* DWM1000 (802.15.4), 6.8 Mbits/s MRF89XAM9A, 200 kbits/s
Microphones	4 SPA2410LR5H-B MEMS Mics Sensitivity: -38 dBV/Pa, SNR 63 dB ADC: 4 ADCS7476, 12-bit, 1 MSPS
Other Interfaces	4 USB ports 10/100 Ethernet port Full HDMI port 3.5mm audio jack Micro SD card slot
Power	5 V, up to 3 W

* Available on the module but not used in this work

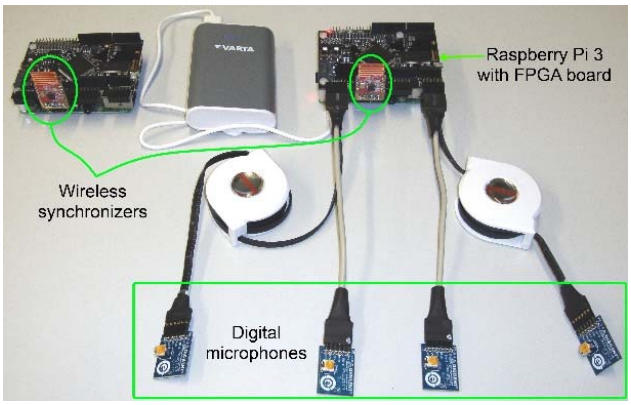

Fig. 2. WHISPER-M4 module connected to 4 microphones and a power bank.

Since we target low-latency applications, we have to use rather small data buffers to minimize the latency of the data acquisition. However, it is difficult to achieve hard real-time requirements for data readout with a small buffer size even in a Linux-based operating system. We found that we need at least 8 ms to reliably read the data from a program running in user-mode on an isolated CPU core. The buffer size for four microphones in this case equals to $0.008 * 4 * F_s$, where F_s is sampling frequency.

The FPGA has two such FIFO buffers. While the FPGA is reading out the data samples from the four microphones and storing them into one buffer, the other buffer is available for reading out from the CPU side. If the CPU has not managed to finish reading out the data within the specified time window, the whole next data packet is dropped to ensure data consistency and prevent overwriting of a partially read buffer with new data. We use polling of the status flag "Empty" in the Wishbone address space (0) to determine when a new data portion is available. Once the Empty flag is cleared, the CPU can issue a burst read of the whole buffer via the Wishbone interface.

The Status Register bits [15:4] represent 12-bit packet counter, and bits [3:0] are the [Empty & Stop & Stalled & Overflow] flags. When the Overflow flag is set, it means that one or more packets were dropped after the current packet. The exact number of the dropped packets can be calculated by comparison of the packet counters of the current and the following packets (see Section IV.B).

C. *Communication and Network*

We investigated two ways of connecting the WHISPER–M4 modules as part of a wireless network using – 1) the 2.4 GHz Wi-Fi (IEEE 802.11n) standard in ad-hoc mode or 2) the DWM1000 transceiver with TDM media access. We excluded the investigation of Bluetooth even though it was available on

the Raspberry Pi. In both cases, there are two topologies for the data transmission graph depending on the algorithm used for

TABLE. II WISHBONE INTERFACE ADDRESS SPACE

Address	*Read*	*Write*
0:2047	Data	N/A
2048	Read FIFO Size	FIFO Reset
2049:2051	0	FIFO Reset
2052:4095	Status Register	FIFO Reset

data processing – a star topology for the centralized algorithm and a full mesh for the distributed processing. When using Wi-Fi, the data transmission graph topology is defined by software.

Although Wi-Fi 802.11n has a high data throughput, the latency of the data transmission is unpredictable and varies a lot with the network load. Specifically, an environment with a lot of access points using the same frequency bands would cause a lot of conflicts thus increasing transmission delays. For real-time audio processing applications, the whole data processing pipeline latency should not exceed ~ 20 to 50 ms [13], therefore the data transmission latency should be in the range of 10 to 20 ms. We measured latencies of wireless data transmission over Wi-Fi in realistic scenario with required data rates and packet sizes (Fig. 3) using a star topology where data are sent from four WHISPER-M4 modules to a PC using UDP protocol. We found that in most of the cases the latencies are unacceptably high, reaching hundreds of milliseconds in some cases. We presume that this happens because of data collisions since all boards may transmit data at the same time. This problem cannot be solved reliably by using 2.4 GHz Wi-Fi to our knowledge. Even though the Wi-Fi data transmission order can be controlled using custom firmware, there are always many other Wi-Fi enabled devices around that operate independently and would interfere in busy environments.

To meet the low-latency requirement, we added the DWM1000 wireless transceiver from DecaWave (IEEE 802.15.4 compliant). This transceiver provides direct control over transmission time allowing implementation of time-division multiplexing (TDM) media access therefore addressing challenges 2) and 3) in Section II. Since WHISPER-M4 modules are synchronized with sub-microsecond precision, we can implement a "circular" data transmission scheme, where each module has its dedicated timeslot for data transmission and all other modules can receive data during this time. This scheme naturally implements the full mesh topology and is ideal for distributed data processing, although centralized processing also can be implemented. In the latter case, one additional WHISPER-M4 module with wired Ethernet connection to the processing unit is required to enable receiving data from four acquisition modules.

TABLE III. DATA TRANSMISSION LATENCIES WITH DWM1000

Packet size (bytes)	*Latency (µs)*
128	318
256	542
512	828
1021	1438

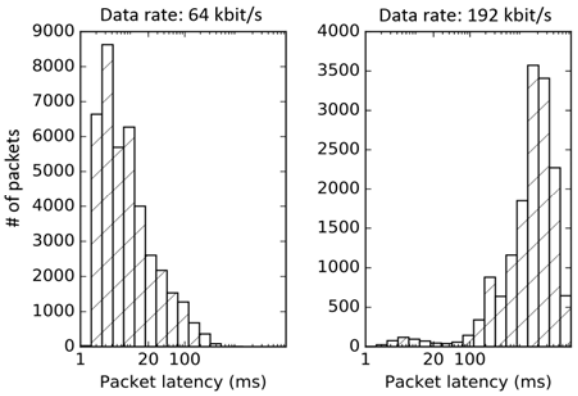

Fig. 3. Wi-Fi latency at different data rates. Four modules are transmitting. Packet size – 256 bytes. Left: 64 kbit/s correspond to 4 microphones sampled at 8 kHz; right: 192 kbit/s correspond to 4 microphones sampled at 24 kHz.

The data transmission latency measurements for the DWM1000 module are shown in TABLE III. When all four WHISPER-M4 modules are used, these latency numbers increase by 4X, but even in this case, these numbers are comparable with the acquisition time for this amount of data.

Even though we evaluated the data transmission with the DWM1000 module, the results presented in the remainder of the paper are obtained by using Wi-Fi 802.11n.

IV. PLATFORM SYNCHRONIZATION

A. *Sampling Clock Synchronization*

The synchronization algorithm is based on a Phase-Locked Loop (PLL) idea. The wireless transceiver MRF89XAM9A transmits the Clock and Data signals to all modules of the WHISPER platform (Fig. 4). The Clock signal recovered from the data stream by the receiver is used to generate the sampling frequency and the Data signal carries information for the packet-level synchronization. Since the transceiver supports a fixed number of data rates, it cannot transmit a Clock signal of an arbitrary frequency. The highest possible data rate 200 kbit/s was chosen because it gives the minimum phase jitter for the Clock signal – about 150 ns.

In order to facilitate the Clock recovery, the data stream has to have as many 0-to-1 and 1-to-0 transitions as possible. The data are structured into packets of 1600 bits that are transmitted continuously. The packets have 13-bit header and 7-bit packet counter, the rest of the packet is filled with alternating 0s and 1s. The packet headers in this case follow with 8 ms intervals and are used to recover from the clock misalignment condition, when one or several clock cycles are missing due to a noise in the radio channel.

In order to generate a phase-locked sampling frequency from the received Clock signal, a digital PLL was implemented in the FPGA. This PLL generates a 48 kHz frequency clock which can be divided by 2, 3 and 6 to get sampling rates of 24 kHz, 16 kHz and 8 kHz. For sampling rates of 44.1 kHz and 22.05 kHz, another PLL has to be used.

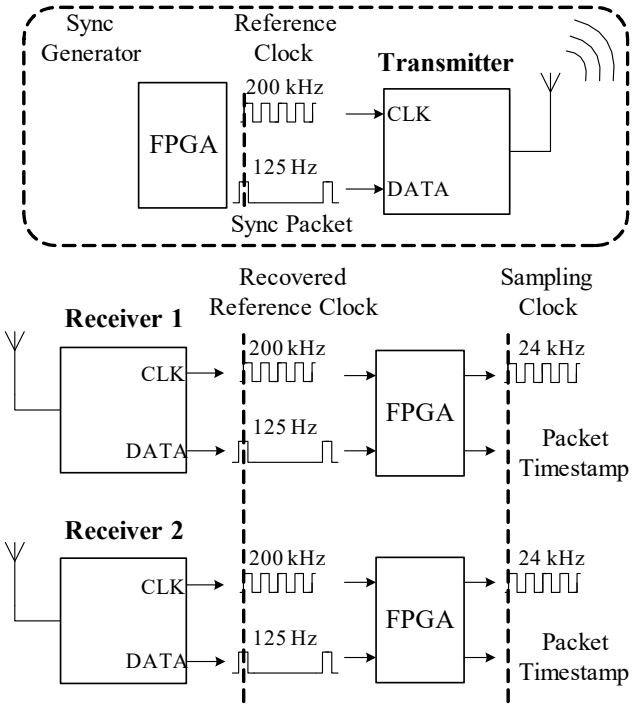

Fig. 4. Wireless synchronization principle.

The sampling rate synchronization is not sufficient to align the received data from two modules. Due to random power-up times, different modules can start filling their data buffers at different times. This gives a constant shift between data samples coming from different modules at the receiving site. To avoid this misalignment, we need to start data acquisition at all modules simultaneously. The "start of acquisition cycle" is sent over the data channel of the synchronizer to all modules. When modules receive this signal they reset their internal time counter, flush buffers and start data acquisition again. Each module counts the clock cycles coming from the synchronizer and embeds this time information into each data packet so that they can be aligned at the receiving site.

B. Packet-Level Synchronisation

By using the synchronization algorithm described in the previous section, the audio can be sampled from all the microphones synchronously. But when we use a wireless network to collect the data from different modules at the processing unit, the data packets might arrive scrambled and some packets could be missing. To address this problem, we developed an algorithm that merges the data coming from the multiple modules and that can deal with packets arriving from different modules in a scrambled order. In this algorithm, packets from all modules will be dropped, if one of them is too slow in transmitting the data.

Every module sends a packet which contains the packet id $p_{id} \in [0, 4096]$ and a module id $w_{id} \in [0, K-1]$ where K is the number of modules.

The main structure we used to collect the data is a FIFO queue implemented with 2 dictionaries. We define the dictionary D^1 as having the following (key, value) pairs

$$D^1: (a, p_{id}) \tag{1}$$

where a is the arrival counter, which is updated every time a new p_{id} is received regardless of its w_{id}. We also define the dictionary D^2 as having the following (key, object) pair

$$D^2: (p_{id}, [(l_0, l_1, \dots, l_{K-1}), n]) \tag{2}$$

where l_k is a byte array with the microphones samples from the module with index k, and n is the number of modules that already sent the packets with the same p_{id}, in other words is the number of non-zero elements in the list $(l_0, l_1, \dots, l_{K-1})$.

This solution with two dictionaries allows us to solve both the problem of having scrambled packets and also the problem of the rollover in the numbering of the packets. The latter is caused by the use of 12 bits to number the packets on the FPGA, thus inducing a rollover in the numbering after every 2^{12} packets. The following examples will clarify the procedure used to ensure that we can process all the received packets correctly.

Let us consider the case where the platform has only two modules and that we start with empty dictionaries. When the first packet arrives, we put the corresponding p_{id} in D^1 with $a = 0$ and we then increment a. We also put an entry in D^2 with the received p_{id} as key. The corresponding object in the entry will have a list of zeros except for the location corresponding to the platform from where the packet originated. In order for this data to be processed, we first need to receive the corresponding p_{id} from the second platform. Only then, with the counter $n = 2$, we have a complete frame. This procedure continues as follows. If we receive a p_{id} that already exists in D^2 we fill the corresponding list with the received data. On the contrary we increase the counter a, insert the p_{id} in D^1 and insert the corresponding data in D^2. Once the FIFO starts filling, the data that is supposed to be processed first is the one which p_{id} is associated with the lowest a in D^1. That is, we process a frame only when the n counter of this entry reaches 2.

In the case we receive a p_{id} which is lower than the lowest p_{id} in D^2, we need to consider that this might be happening because of the rollover of the packet counter. In the case we already received data from that module with a p_{id} which is higher than the one we received, it means that rollover has happened and we keep the packet. On the contrary, it means that the packet is too late and the data coming from the other module with the same p_{id} has already been discarded.

V. BEAMFORMING ALGORITHM

This section describes the implementation of a particular beamforming algorithm along with a version of this algorithm that can be implemented in real time.

The Linearly Constrained Minimum Variance (LCMV) beamformer is a widely used beamforming algorithm for both speech enhancement and dereverberation [14][15]. It is a spatial filtering method that localizes the sources based on the data recorded at different locations and in our case, this data

come from the multiple microphones placed at these locations. The basic idea of LCMV consists in minimizing the power of the filtered signal obtained by summing the signal from each microphone after applying meaningful delays, with a constraint. This constraint forces the filter to output power from a specific location and the power minimization allows one to suppress the signals coming from undesired locations.

Given a signal $x(t) \in \mathbb{R}^M$ where M is the number of microphones used in the array, we define an optimal set of weights $\widehat{w}_{opt}$ which will be applied to the input signal in order to provide a denoised signal $y(t) \in \mathbb{R}$. We then define $\boldsymbol{C}$ as the $M \times Q$ constraint matrix for LCMV, where Q is the number of desired constraints. The constraints are defined as the direction of a source with respect to each microphone. If we define $d(\theta_k)$ as the set of Difference of Arrival (DOA) in phase between source k and all the microphones, we can define

$$C = [d(\theta_1), d(\theta_2), \dots, d(\theta_Q)] \tag{3}$$

as the complex vector representing the position of the source of interest. Then we define the response vector f which is a binary vector representing the desired response of the beamformer to the selected sources in the constraint matrix. A value of 1 will mean that the source is enhanced while a value of 0 will mean that the source is suppressed. In our case, we focus on only defining one constraint on one source and suppressing all the others. In this setup, our formulation will result in a Minimum Variance Distortion Ratio (MVDR)[16] which is a particular case of LCMV. The optimization problem of LCMV is shown in (4) and the solution is shown in (5):

$$\widehat{w}_{opt} = arg\min_{w} w^H \boldsymbol{R}_{xx} w \qquad s.t. C^H w = f \tag{4}$$

$$\widehat{w}_{opt} = \boldsymbol{R}_{xx}^{-1} C (C^H \boldsymbol{R}_{xx}^{-1} C)^{-1} f \tag{5}$$

where $\widehat{w}_{opt}$ are optimal weights, $\boldsymbol{R}_{xx}$ is the covariance matrix of the microphone recordings defined as $\boldsymbol{R}_{xx} = E\{xx^H\}$, with $E\{\cdot\}$ representing the expected value over the signal in the time window. One of the assumptions of this algorithm is that the relative positions of the microphones and the source are known. However, we want to relax this constraint in this work, and be able to run the algorithm without any a priori knowledge of the positions of the microphones and the source. In order to make this approach blind, a necessary estimation step of the source position is needed. As shown in [16], it is possible to estimate the forward mapping (also known as mixing matrix) of a source with respect to the microphones by solving an eigendecomposition problem over the covariance matrix of the signals recorded when only the source of interest is active.

$$[\boldsymbol{U}, \boldsymbol{S}] = eig(\boldsymbol{R}_{xx}) \tag{6}$$

where $\boldsymbol{U}$ is the matrix of eigenvectors and $\boldsymbol{S}$ is the diagonal matrix of eigenvalues. The C is computed by taking the eigenvector corresponding to the maximum eigenvalue.

A. Real-time Operation

We benchmarked the algorithm on a smartphone (LG Nexus 5, Quad-core 2.3 GHz, 2GB RAM, running Ubuntu Touch OS) in order to evaluate the performance for real-time operation. In this case, real-time is achieved if the time taken for the algorithm to process a frame of data, T_{comp}, is less than the frame acquisition time N_S/F_S, where N_S is the number of samples in the frame and F_S is the sampling frequency. In this way, the algorithm can keep up with new data coming in.

Because we process the data in frames of N_s samples, there is a trade-off of processing time versus the window size of the data processed by the algorithm at every time step. On one hand, the window should be as large as possible, to ensure the quality and distortionless output of the filtered signal and to have enough time to process the data. On the other hand, a large window means that a larger delay is introduced in the pipeline that contradicts low latency requirements. This consideration for real-time performance does not yet include the additional delay that will be introduced by the wireless transmission of the results from the platform to the central processing unit as previously discussed in Section III.C. The LCMV optimization is based on the calculation of the covariance matrix of the microphone outputs which is assumed to be full rank and well-conditioned. When computing LCMV offline, the covariance matrix is estimated by taking into account the entire signal and this guarantees full rank. In the online scenario, the entire signal is not available, therefore the covariance matrix has to be estimated over time. In order to guarantee full rank, we need to do a leaky update of the covariance matrix and to include samples from past windows. In this way, the covariance matrix is well-conditioned and updated on the kth step as:

$$\widetilde{\boldsymbol{R}}_{xx}(k) = \beta \widetilde{\boldsymbol{R}}_{xx}(k-1) + (1-\beta) x(k) x^H(k) \tag{9}$$

where the β parameter controls how much of the new covariance matrix is integrated in the current estimate $\widetilde{\boldsymbol{R}}_{xx}(k)$. Overall, the computational complexity of this algorithm is dominated by the large number of matrix inversions ($N_s M^2$) of complexity $O(M^3)$ that have to be computed. This might represent an issue for real-time performance.

B. Real-time Implementation

In order to overcome this problem, we derived a gradient descent (GD) update of LCMV that consists of simple matrix multiplications. The GD LCMV has a lower computational cost than matrix inversion [17]. The update can be derived by using a Lagrange multiplier λ to include the constraint in the cost function. This way we can convert the constraint optimization problem in (4) into the unconstrained one (10).

$$\underset{w}{argmin} \frac{1}{2} w^H \boldsymbol{R}_{xx} w + Real\{\lambda^H (C^H w - f)\} \tag{10}$$

The equivalence of (10) and (4) depends on the correct choice of λ. In order to minimize (10), a gradient descent approach can be used as described in (11) to (13) where n is the current update step:

$$w(n+1) = \boldsymbol{P}\left(I - \mu \widetilde{\boldsymbol{R}}_{xx}\right) w(n) + G \tag{11}$$

$$\boldsymbol{P} = [I - C(C^H C)^{-1} C^H] \quad (12)$$

$$G = C(C^H C)^{-1} f \quad (13)$$

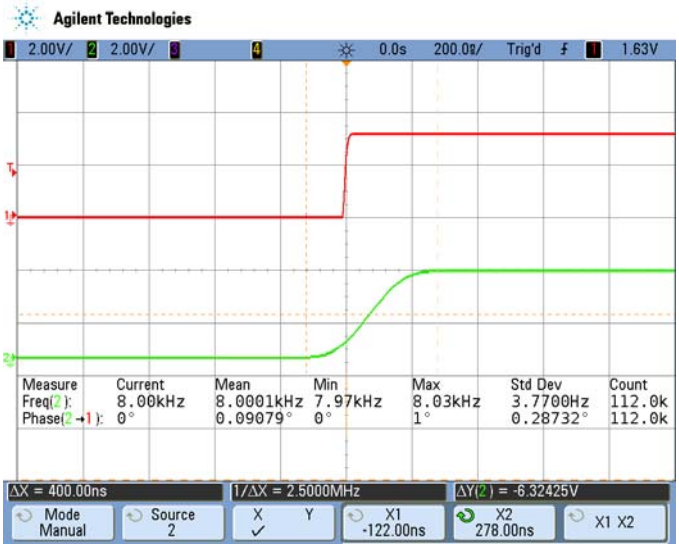

Fig. 5. Synchronization measurement results. The red (top) curve shows the rising edge of the sampling clock measured at one module and the green (bottom) curve shows the same measurement at another module averaged over 16,384 samples that gives an approximate cumulative distribution function of the second rising edge with respect to the first one. Mean phase shift 0.091˚ equals to 32 ns at 8 kHz, Phase jitter (Std Dev) = 0.287˚ equals to 100 ns at 8 kHz.

VI. Results

Phase synchronization measurements from two WHISPER-M4 modules are shown in Fig. 5. The constant phase shift between the two modules depends on the relative spatial positions of the modules and is caused by the multipath radio wave propagation. The observed phase shift of 32 ns corresponds to a negligible spatial shift (much less than 1 mm) when sampling the acoustic signal in the air, so this value does not affect our processing algorithm. The second curve (green) shows the approximate cumulative distribution function of the sampling clock rising edge of the second module with respect to the first one.

For real time performance, we require that the computation of LCMV is completed before the arrival of the next window of data. As we can see from Fig. 6, the dashed line represents the limit of computation time to achieve real time performance at a sampling rate of 8 kHz and 256 samples in a window. While LCMV has a computational complexity which does not allow for real time operation, GD LCMV assures that the beamforming algorithm can be done in time, before the next window of data is received by the central platform.

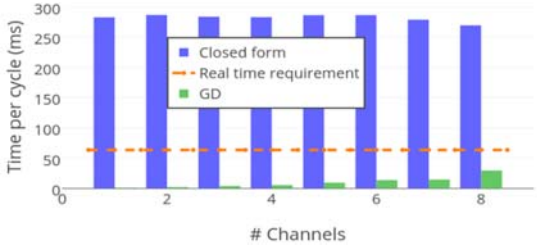

Fig. 6. Comparison of LCMV and GD LCMV performance and realtime requirement for different number of microphone channels.

Next, we report results from the different beamforming algorithms on two datasets: the Telluride Neuromorphic Dataset and the WHISPER conference room dataset. We report first on the Telluride dataset which was recorded during an experiment carried out at the Telluride Neuromorphic Cognition Engineering Workshop in 2016. The data was collected from eight synchronized microphones (DPA 4060BM) in a semi-cluttered room. The eight microphones were placed in an approximate circular configuration, with each pair of microphones placed 20 cm from each other. We played two audiobooks from two loudspeakers. These recordings are noisy due to the reverberation conditions of this semi-cluttered room with the two sources at +/- 60 degrees with respect to the bottom microphones.

We evaluate the performance of the LCMV beamformer variants using two metrics: the signal-to-interference (SIR) metric, a classical evaluation measure for blind source separation [18] and the STOI, a measure of speech intelligibility. The SIR value is calculated as described in [18] and in (14). For this, we need to provide both the separated signals $y_{target} = w_{target}x$ and $y_{interf} = w_{interf}x$ which are obtained by estimating speaker dependent weights w_{target} and w_{interf},

$$SIR = 10 \log_{10} \frac{\|y_{target}\|^2}{\|y_{interf}\|^2} \quad (14)$$

The STOI metric evaluates the quality of a speech signal with respect to ground truth. The score ranges from 0 to 1 where 1 corresponds to a perfect separation (or denoising) in which the separated signal from a mixture is as intelligible as the original signal.

Fig. 7 shows the STOI results for the mixture and the separated sources using both the closed form solution of the LCMV algorithm and the GD LCMV algorithm on this Telluride dataset. The results show that the STOI scores for both the closed form and the GD version are very similar for three different window size samples thus proving that we can use the GD LCMV in practice.

The dataset for the conference room dataset was collected using two WHISPER-M4 modules placed on a conference table in front of two loudspeakers as shown in Fig. 8.

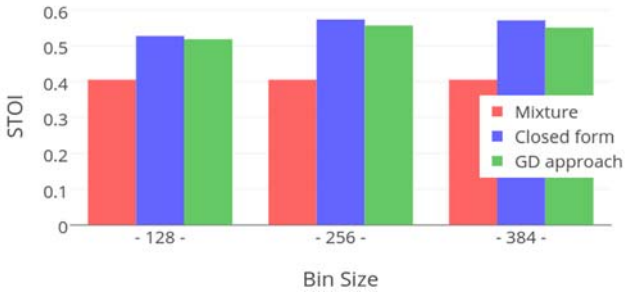

Fig. 7. STOI results of LCMV and GD LCMV on the Telluride dataset.

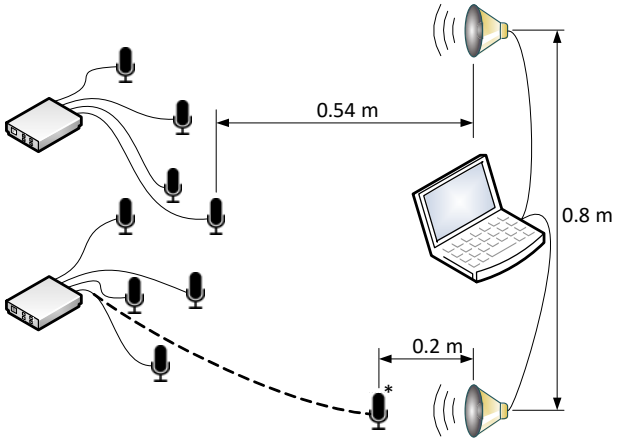

Fig. 8. Experimental setup for the WHISPER dataset. Approximate relative positions of speakers and microphones are shown. Data are transmitted to the laptop wirelessly via Wi-Fi. The *-marked mic position is used for "10 loud" and "10 soft" experiments.

This configuration also had eight microphones, four from each module. The recordings were done over a period of 1 min with either two male or two female speakers reading audiobooks from different loudspeakers. To investigate effects of signal-to-noise ratio (SNR), we kept the volume of one speaker constant while we varied the volume of the second speaker. We also investigated the effect of placing a microphone close to either the desired source or the interfering source. We kept the SNR at 10dB and recorded a scenario where one of the microphones was 20 cm from the loud source and a second scenario where the microphone was 20 cm from the soft source. Additionally, we recorded each speaker alone for 15 seconds in order to estimate the constraint matrix C as described in Section V.

The beamforming algorithm was applied in real-time on the data recorded from two modules when the microphone sampling frequencies were either synchronized or not synchronized across the modules. Fig. 9 and Fig. 10 show the beamforming quality through SIR and STOI scores, of the separated signals when using either the synchronized or non-synchronized samples of the audio mixture recorded from the two modules in the WHISPER conference room dataset.

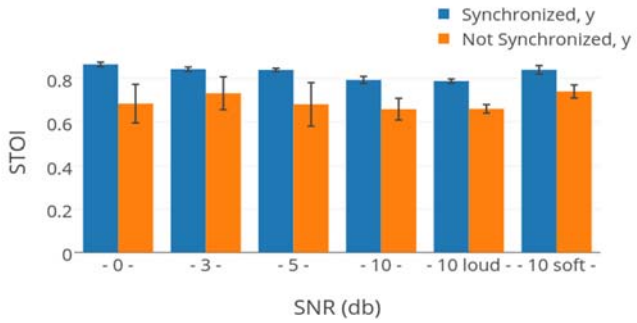

Fig. 9. STOI scores computed from synchronized and non-synchronized samples for four different signal-to-noise ratios of the microphone placement shown in Fig. 8 and by placing a microphone close to the speaker with the louder source ("10 loud"), and then to the speaker with the softer source ("10 soft").

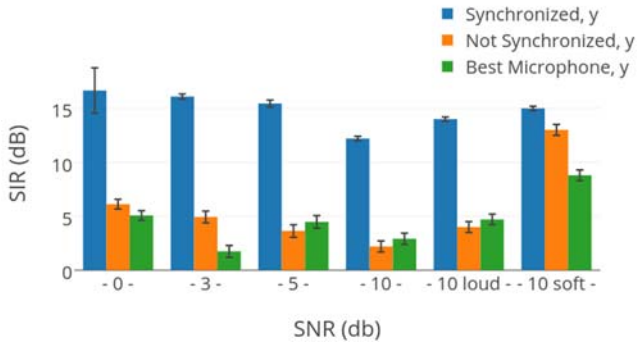

Fig. 10. SIR scores computed from samples which are synchronized and non-synchronized for four different signal-to-noise ratios, compared with SIR obtained from the best microphone, and by placing a microphone close to the speaker with the louder source ("10 loud"), and then to the speaker with the softer source ("10 soft").

In particular, Fig. 9 shows that the average STOI score for synchronized samples is almost 20% higher than the score for non-synchronized samples. Fig. 10 shows that the SIR scores for non-synchronized samples are not statistically better than the SIR obtained by selecting the microphone with the best SNR out of all the eight microphones. Both figures show the degradation in the quality of the separated source when the samples are not synchronized. Additionally, we can see that having a microphone very close to either the desired source or the interfering source is beneficial for the beamforming. In particular having a microphone very close to the desired source, which is 10 dB softer then the interfering source increases the STOI by 3% and the SIR by 15%.

VII. Conclusion

This work describes a distributed portable multi-microphone platform with capabilities for acting as nodes in a wireless acoustic sensor network. This platform addresses three main challenges for use in a WASN. The first is the synchronization of the sampling clocks across different modules with a relative clock jitter of less than 300ns (3 standard deviations). The second is the synchronization of data packets received from multiple modules and the third is

the development of a wireless transmission system that will ensure low latency transmission between the nodes of the platform. Our beamforming results from the microphone samples in the WHISPER dataset demonstrate that this tight synchronization of the sampling clock on all modules is important for obtaining the best beamforming results and that beamforming with this platform gave SNR results that are in line with the expected outcome of the algorithm. The platform can be miniaturized in the future to further reduce both the latency and power consumption, and keeping only the features that are needed for the proposed solutions in this work.

ACKNOWLEDGMENT

We acknowledge members of the Sensors group at the Institute of Neuroinformatics, Adrian Huber for proofreading the article, and Matthew Rahtz for the WiFi measurements on the WHISPER platform.

REFERENCES

[1] J Benesty, J Chen, Y Huang, and J Dmochowski, "On microphone-array beamforming from a MIMO acoustic signal processing perspective," *IEEE Trans. Audio, Speech, and Language Processing*, 15 (3), 1053-1065, 2007.

[2] D.Y. Levin, E.A.P. Habets and S. Gannot, "On the average directivity factor attainable with a beamformer incorporating null constraints," *IEEE Signal Processing Letters*, 22 (11), 2122-2126, 2015.

[3] A. Bertrand, S. Doclo, S. Gannot, N. Ono, and T. van Waterschoot. Guest editorial: Special issue on wireless acoustic sensor networks and ad hoc microphone arrays. *Signal Processing*, 107, pp.1-3, 2015.

[4] S. Markovich-Golan, S. Gannot and I. Cohen, "Distributed multiple constraints generalized sidelobe canceler for fully connected wireless acoustic sensor networks," *IEEE Trans. Audio, Speech, and Language Processing*, 21 (2), pp. 343-356, 2013.

[5] A. Bertrand, "Applications and trends in wireless acoustic sensor networks: A signal processing perspective,", *2011 18th IEEE Symposium on Communications and Vehicular Technology in the Benelux (SCVT),* Ghent, pp. 1-6, 2011.

[6] S. Miyabe, N. Ono and S. Makino, "Blind compensation of interchannel sampling frequency mismatch for ad hoc microphone array based on maximum likelihood estimation," Signal Processing, 107, pp. 185–196, 2015.

[7] L. Girod, M. Lukac, V. Trifa, and D. Estrin, "The design and implementation of a self-calibrating distributed acoustic sensing platform," Proc. of the 4th International Conference on Embedded Networked Sensor Systems (SenSys '06), ACM, New York, NY, USA, pp. 71-84, 2016.

[8] C. Schörkhuber, M. Zaunschirm, and I. H. Zmölnig, "WiLMA-Wireless Largescale Microphone Array," Proc. of the Linux Audio Conference, Apr. 2014.

[9] 3D AudioSense, http://www.3daudiosense.com/blog/3d-audiosense-beaglebone-black-cape.

[10] K. Ochi, N. Ono, S. Miyabe, and S. Makino, "Multi-talker speech recognition based on blind source separation with adhoc microphone array using smartphones and cloud storage," Proc. of 2016 IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), 107, pp. 185–196, 2016.

[11] R. Lienhart, I. Kozintsev, M. Yeung and S. Wehr, "On the importance of exact synchronization for distributed audio signal processing," Proc. of 2003 IEEE Int. Conf. Acoustics, Speech, and Signal Processing (ICASSP), pp. IV-840-843, 2013.

[12] A. Winkler, M. Latzel, and I. Holube. "Open versus closed hearing-aid fittings: a literature review of both fitting approaches," Trends in Hearing, 2016.

[13] M. Keetels and J. Vroomen, "Perception of synchrony between the senses," Chapter 9 in The Neural Bases of Multisensory Processes, M.M. Murray and M.T. Wallace, editors, Boca Raton (FL), CRC Press/Taylor & Francis, 2012.

[14] O. L. Frost, III, "An algorithm for linearly constrained adaptive array processing", Proceedings of the IEEE, 60 (8), pp. 926–935, August 1972

[15] B. D. Van Veen, W. Van Drongelen, M. Yuchtman, and A. Suzuki, "Localization of brain electrical activity via linearly constrained minimum variance spatial filtering," IEEE Transactions on Biomedical Engineering, 44 (9), pp. 867–880, 1997.

[16] L. Parra and P. Sajda, "Blind source separation via generalized eigenvalue decomposition," J. Mach. Learn. Res. 4, pp. 1261–1269, 2003.

[17] X. Guo, L. Chu, and B. Li, "Robust adaptive LCMV beamformer based on an iterative suboptimal solution," Radioengineering, 24(2), pp. 572–582, 2015.

[18] E. Vincent, R. Gribonval and C. Févotte, "Performance measurement in blind audio source separation," *IEEE Trans. Audio, Speech and Language Processing*, 14(4), pp 1462-1469, 2006.